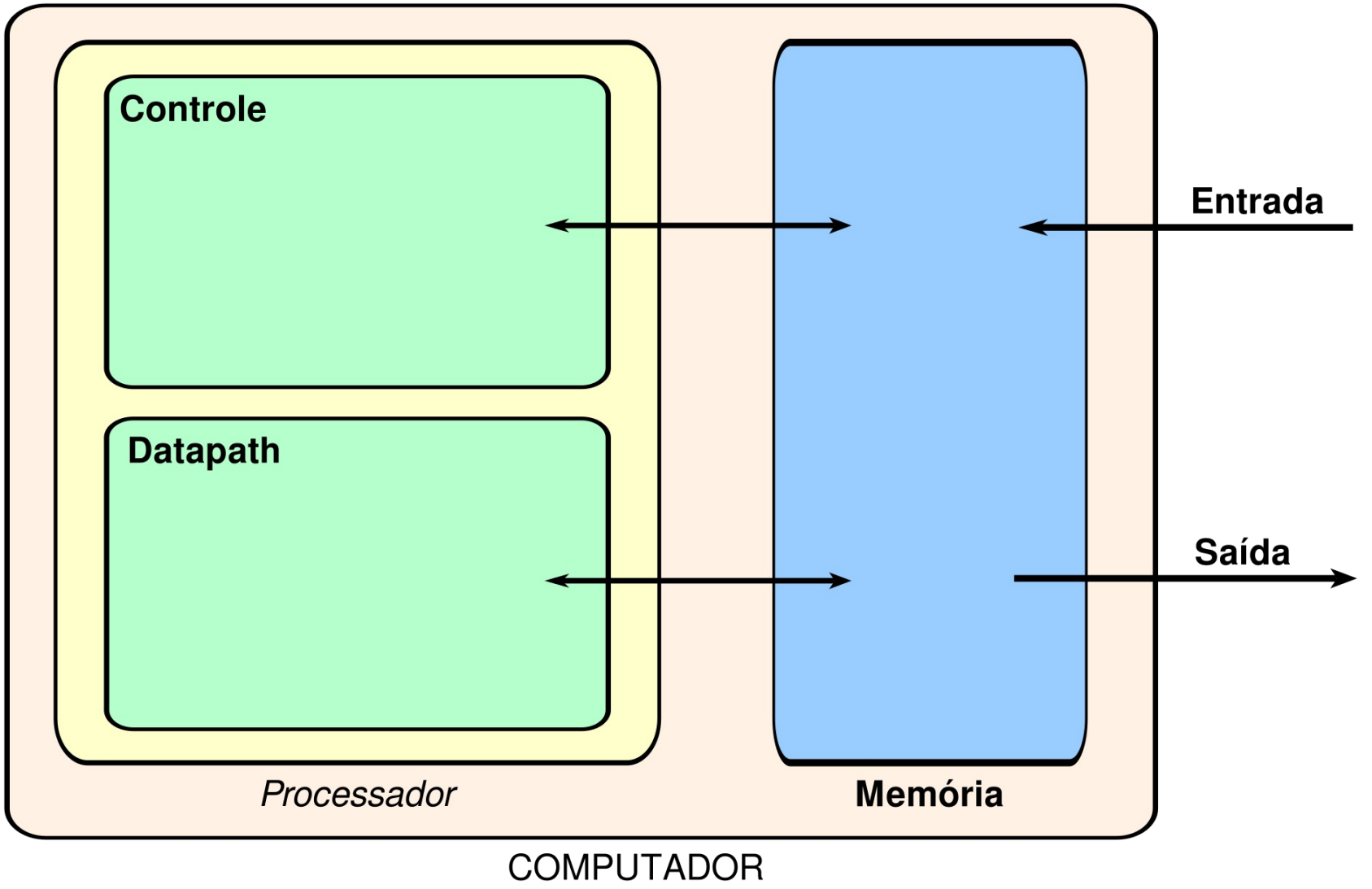
MOTIVAÇÃO


Controle
Datapath
Entrada
Saída
Processador
Memória
COMPUTADOR

MOTIVAÇÃO


Servo ou magia sem controle?
int não é Z; float não é R
Olá, mundo!
Memória importa
Entrada/Saída/Rede
Motivação
Evolução
Rito de passagem
Pilha tecnológica

Olá, mundo!

```c
/**
 * CR6.190A: Arquitetura de Computadores
 * https://cursos.computacaoraiz.com.br
 *
 * ola.c
 * O canônico "Olá, mundo!" da programação.
 */

#include <stdio.h>

int main(void)
{
    printf("%s\n", "Olá, mundo!");

    return 0;
}
```

← interação hardware/software em diversos níveis; fundamental para o entendimento da computação!

```
[abrantesasf@cosmos ~/cr6.190a/introducao]$ ./ola
Olá, mundo!
```

Olá, mundo!

```
[abrantesasf@cosmos ~/cr6.190a/introducao]$ xxd -b -c 13 -g 0 ola > ola1.txt
[abrantesasf@cosmos ~/cr6.190a/introducao]$ cut -c 11-114 ola1.txt > ola2.txt
[abrantesasf@cosmos ~/cr6.190a/introducao]$ tr -d "\n" < ola2.txt > ola3.txt
```

1 (1 of 7) Fit Width

A4, margens de 0,5 cm, fonte tamanho 6 pt: 7 páginas de 0s e 1s (127.680).

```c
/**
 * CR6.190A: Arquitetura de Computadores
 * https://cursos.computacaoraiz.com.br
 *
 * ola.c
 * O canônico "Olá, mundo!" da programação.
 */

#include <stdio.h>

int main(void)
{
    printf("%s\n", "Olá, mundo!");

    return 0;
}
```

Olá, mundo!

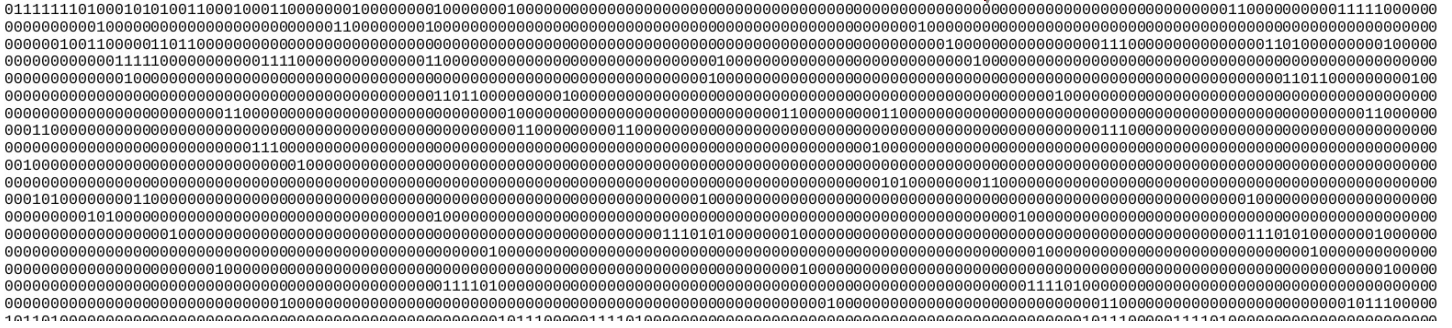
```
[abrantesasf@cosmos ~/cr6.190a/introducao]$ objdump -s ola | grep section
Contents of section .interp:
Contents of section .note.gnu.property:
Contents of section .note.gnu.build-id:
Contents of section .note.ABI-tag:
Contents of section .gnu.hash:
Contents of section .dynsym:
Contents of section .dynstr:
Contents of section .gnu.version:
Contents of section .gnu.version_r:
Contents of section .rela.dyn:
Contents of section .rela.plt:
Contents of section .init:
Contents of section .plt:
Contents of section .plt.got:
Contents of section .plt.sec:
Contents of section .text:
Contents of section .fini:
Contents of section .rodata:
Contents of section .eh_frame_hdr:
Contents of section .eh_frame:
Contents of section .init_array:
Contents of section .fini_array:
Contents of section .dynamic:
Contents of section .got:
Contents of section .data:
Contents of section .comment:
```

```
1 (1 of 1) [abrantesasf@cosmos ~/cr6.190a/introducao]$ objdump -S ola > ola_objdump.txt

Disassembly of section .init:
0000000000001000 <_init>:
    1000: f3 0f 1e fa                   endbr64
    1004: 48 83 ec 08                   sub    $0x8,%rsp
    1008: 48 8b 05 d9 2f 00 00          mov    0x2fd9(%rip),%rax        # 3fe8 <__gmon_start__@Base>
    100f: 48 85 c0                      test   %rax,%rax
    1012: 74 02                         je     1016 <_init+0x16>
    1014: ff d0                         call   *%rax
    1016: 48 83 c4 08                   add    $0x8,%rsp
    101a: c3                            ret

Disassembly of section .plt:
0000000000001020 <.plt>:
    1020: ff 35 9a 2f 00 00             push   0x2f9a(%rip)        # 3fc0 <_GLOBAL_OFFSET_TABLE_+0x8>
    1026: f2 ff 25 9b 2f 00 00          bnd jmp *0x2f9b(%rip)        # 3fc8 <_GLOBAL_OFFSET_TABLE_+0x10>
    102d: 0f 1f 00                      nopl   (%rax)
    1030: f3 0f 1e fa                   endbr64
    1034: 68 00 00 00 00                push   $0x0
    1039: f2 e9 e1 ff ff ff             bnd jmp 1020 <_init+0x20>
    103f: 90                            nop

Disassembly of section .plt.got:
0000000000001040 <__cxa_finalize@plt>:
    1040: f3 0f 1e fa                   endbr64
    1044: f2 ff 25 ad 2f 00 00          bnd jmp *0x2fad(%rip)        # 3ff8 <__cxa_finalize@GLIBC_2.2.5>
    104b: 0f 1f 44 00 00                nopl   0x0(%rax,%rax,1)

Disassembly of section .plt.sec:
0000000000001050 <puts@plt>:
    1050: f3 0f 1e fa                   endbr64
    1054: f2 ff 25 75 2f 00 00          bnd jmp *0x2f75(%rip)        # 3fd0 <puts@GLIBC_2.2.5>
    105b: 0f 1f 44 00 00                nopl   0x0(%rax,%rax,1)

Disassembly of section .text:
0000000000001060 <_start>:
    1060: f3 0f 1e fa                   endbr64
    1064: 31 ed                         xor    %ebp,%ebp
    1066: 49 89 d1                      mov    %rdx,%r9
    1069: 5e                            pop    %rsi
    106a: 48 89 e2                      mov    %rsp,%rdx
    106d: 48 83 e4 f0                   and    $0xfffffffffffffff0,%rsp
    1071: 50                            push   %rax
    1072: 54                            push   %rsp
    1073: 45 31 c0                      xor    %r8d,%r8d
    1076: 31 c9                         xor    %ecx,%ecx
    1078: 48 8d 3d ca 00 00 00          lea    0xca(%rip),%rdi        # 1149 <main>
    107f: ff 15 53 2f 00 00             call   *0x2f53(%rip)        # 3fd8 <__libc_start_main@GLIBC_2.34>
    1085: f4                            hlt
    1086: 66 2e 0f 1f 84 00 00          cs nopw 0x0(%rax,%rax,1)
```

Olá, mundo!

```c
/**
 * CR6.190A: Arquitetura de Computadores
 * https://cursos.computacaoraiz.com.br
 *
 * ola.c
 * O canônico "Olá, mundo!" da programação.
 */

#include <stdio.h>

int main(void)
{
    printf("%s\n", "Olá, mundo!");

    return 0;
}
```

```
1 (1 of 1)  Fit Width

Disassembly of section .init:
0000000000001000 <_init>:
    1000: f3 0f 1e fa                endbr64
    1004: 48 83 ec 08                sub    $0x8,%rsp
    1008: 48 8b 05 d9 2f 00 00       mov    0x2fd9(%rip),%rax        # 3fe8 <__gmon_start__@Base>
    100f: 48 85 c0                   test   %rax,%rax
    1012: 74 02                      je     1016 <_init+0x16>
    1014: ff d0                      call   *%rax
    1016: 48 83 c4 08                add    $0x8,%rsp
    101a: c3                         ret

Disassembly of section .plt:
0000000000001020 <.plt>:
    1020: ff 35 9a 2f 00 00          push   0x2f9a(%rip)        # 3fc0 <_GLOBAL_OFFSET_TABLE_+0x8>
    1026: f2 ff 25 9b 2f 00 00       bnd jmp *0x2f9b(%rip)        # 3fc8 <_GLOBAL_OFFSET_TABLE_+0x10>
    102d: 0f 1f 00                   nopl   (%rax)
    1030: f3 0f 1e fa                endbr64
    1034: 68 00 00 00 00             push   $0x0
    1039: f2 e9 e1 ff ff ff          bnd jmp 1020 <_init+0x20>
    103f: 90                         nop

Disassembly of section .plt.got:
0000000000001040 <__cxa_finalize@plt>:
    1040: f3 0f 1e fa                endbr64
    1044: f2 ff 25 ad 2f 00 00       bnd jmp *0x2fad(%rip)        # 3ff8 <__cxa_finalize@GLIBC_2.2.5>
    104b: 0f 1f 44 00 00             nopl   0x0(%rax,%rax,1)

Disassembly of section .plt.sec:
0000000000001050 <puts@plt>:
    1050: f3 0f 1e fa                endbr64
    1054: f2 ff 25 75 2f 00 00       bnd jmp *0x2f75(%rip)        # 3fd0 <puts@GLIBC_2.2.5>
    105b: 0f 1f 44 00 00             nopl   0x0(%rax,%rax,1)

Disassembly of section .text:
0000000000001060 <_start>:
    1060: f3 0f 1e fa                endbr64
    1064: 31 ed                      xor    %ebp,%ebp
    1066: 49 89 d1                   mov    %rdx,%r9
    1069: 5e                         pop    %rsi
    106a: 48 89 e2                   mov    %rsp,%rdx
    106d: 48 83 e4 f0                and    $0xfffffffffffffff0,%rsp
    1071: 50                         push   %rax
    1072: 54                         push   %rsp
    1073: 45 31 c0                   xor    %r8d,%r8d
    1076: 31 c9                      xor    %ecx,%ecx
    1078: 48 8d 3d ca 00 00 00       lea    0xca(%rip),%rdi        # 1149 <main>
    107f: ff 15 53 2f 00 00          call   *0x2f53(%rip)        # 3fd8 <__libc_start_main@GLIBC_2.34>
    1085: f4                         hlt
    1086: 66 2e 0f 1f 84 00 00       cs nopw 0x0(%rax,%rax,1)
```



2ª verdade nua e crua: você tem que saber assembly!

- Provavelmente você nunca escreverá um programa em assembly:
    - compiladores são melhores do que você

- Mas SABER assembly é uma habilidade FUNDAMENTAL para entender:
    - como os os computadores funcionam
    - como o seu código em linguagem de alto nível C é transformado em binário executável
    - o comportamento de programas na presença de bugs
    - ajustes de performance:
        - otimizações feitas/não feitas pelo compilador
        - entender fontes de ineficiências nos programas
    - implementar softwares de sistemas:
        - compiladores (o alvo é código de máquina)
        - sistemas operacionais
    - criar/combater malwares

- RISC-V 32 bits